倍精度に基づく四倍精度四則演算法の誤差とその応用 (科学技術計算における理論と応用の新展開)

AUTHOR(S):

山中, 脩也; 大石, 進一

CITATION:

山中, 脩也 ...[et al]. 倍精度に基づく四倍精度四則演算法の誤差とその応用 (科学技術計算における理論と応用の新展開). 数理解析研究所講究録 2012, 1791: 216-225

ISSUE DATE:

2012-04

URL:

http://hdl.handle.net/2433/172821

数理解析研究所講究録
第1791巻 2012年 216-225

倍精度に基づく四倍精度四則演算法の誤差とその応用

早稲田大学 理工学研究所 山中 脩也 (Naoya Yamanaka) *
Research Institute for Science and Engineering, Waseda University
早稲田大学 理工学術院, JST / CREST 大石 進一 (Shin'ichi Oishi)†

1 はじめに

現在の数値計算では倍精度演算が広く利用されている．しかし時にそれより高精度な演算が必要となることがある．それらを達成するひとつの方法は任意精度の浮動小数点演算を用いることである．任意精度を達成するソフトウエアとしてMPFR [1] やARPREC [2] が広く知られている．

一方，任意精度ほどの高精度は必要なく，倍精度数より少し高精度にするだけで十分なこともある．倍精度演算を利用した四倍精度や八倍精度の計算法としてHida, Li, Baileyが考案したQD/DD [3] が広く知られている．QD/DDは四倍精度や八倍精度に特化しており，任意精度の計算法に比べて高速に計算できるという特徴を持つ．本報告の前半では倍精度演算に基づく四倍精度四則演算法とその誤差について述べる．特に加減算と乗算は計算機上で高速に計算可能な誤差上限を与える．また，本報告の後半では，前半部に示した四倍精度乗算法とその誤差上限を用いて，指数関数の高精度で高信頼かつ高可搬な計算法について述べる．

初等関数近似計算手法は古くから研究され多くの計算法が存在する．現在の浮動小数点数を用いた計算技術は大変高速であり，また，精度の面でもほとんどの入力に対して高精度な結果を計算できる．一方で，結果の精度が保証された初等関数の計算技術の構築は比較的容易ではなく，精度や実行時間の面で課題も多い．この現状を踏まえ，本報告では倍精度浮動小数点数を用いた高精度・高信頼・高可搬な指数関数計算法について述べる．本報告における高精度な計算法とは，倍精度浮動小数点数のみを用いて約四倍精度浮動小数点数程度の結果を返す計算法のことを指す．ここで四倍精度とは，倍精度浮動小数点数の倍の精度のことを指すが，IEEE 754-2008で規格化されたbinary128と比較すると若干精度が落ちるものである．なお，高精度な計算法は倍精度浮動小数点数とその演算のみを用いて計算を行ない，内部で擬似的な多倍長演算を用いて四倍精度の結果を求めるため，ハードウェアの変更は必要としない．本報告で高信頼な計算法とは，丸め誤差や打ち切り誤差などの数値計算の工程で発生する全ての誤差を考慮し，数学的に正しい結果を数値計算によって導く計算法を指す．これにより100%誤りの無い解が得られる．最後に，本報告で高可搬な計算法とは，倍精度浮動小数点システムにおいて四則演算の計算結果を最近点に丸めることができれば，所望の計算結果を与えることができる計算法を指す．高可搬な計算法を構築するのに必要な状況は，広く利用されている浮動小数点数規格に基づいていれば十分であり，現在のほぼ全ての計算機で実装されている．

以上より，本報告は現在使っている計算機環境のまま，僅かなソフトウエアの変更によって，四倍精度の結果を精度保証付きで得ることができ，100%誤りの無い解が得られる計算手法を創造することを目的としている．なお本報告を通して，倍精度形式と倍精度演算がIEEE 754-1985 標準規格に基づくことを仮定する．$fl(\cdot)$ は括弧の中を最近点丸めを用いて倍精度演算することと定義する．$\mathbb{F}$ は倍精度浮動小数点数全体の集合とする．実数 $c \in \mathbb{R}$ に対する $\mathrm{ulp}(c)$ は次のように定義される：

$$\mathrm{ulp}(c) = 2^{e-52}, \text{iff } |c| \in [2^e, 2^{e+1}) \tag{1}$$

*naoya_yamanaka@suou.waseda.jp
†Faculty of Science and Engineering, Waseda University & JST/CREST

ただし，e は整数である．なお，本報告を通してマシンイプシロン $\mathrm{eps} = 2^{-53}$ と定義する．

2 無誤差変換

2.1 加乗算に関する無誤差変換

Knuth は 2 つの浮動小数点数 a, b の加算は，誤差なく計算できることを示した（**Algorithm 1**）[4]．また，もし 2 つの浮動小数点数 a, b の間に $|a| \geqq |b|$ という関係がある時はより高速に計算を行なうことができる（**Algorithm 2**）[4]．

Algorithm 1 加算の無誤差変換 TwoSum

加算に関する無誤差変換．このとき

$$a + b = x + y \tag{2}$$

$$|y| \leqq \frac{1}{2}\,\mathrm{ulp}\,(x) \tag{3}$$

が成立する．

function $[x, y] =$ TwoSum(a, b)
$x = fl(a + b)$
$z = fl(x - a)$
$y = fl((z - (x - z)) + (b - z))$
end

Algorithm 2 加算の無誤差変換 FastTwoSum

加算に関する無誤差変換．ただし，$|a| \geqq |b|$ が成立していることを仮定する．このとき

$$a + b = x + y \tag{4}$$

$$|y| \leqq \frac{1}{2}\,\mathrm{ulp}\,(x) \tag{5}$$

が成立する．

function $[x, y] =$ FastTwoSum(a, b)
$x = fl(a + b)$
$y = fl(b - (x - a))$
end

Algorithm 1 と **Algorithm 2** は，下記の事実を示している：

- x は $a + b$ の最も良い近似である．
- x の誤差 $y = a + b - x$ は浮動小数点数である．
- y は上記の手法を用いて浮動小数点演算のみで計算できる．

乗算に対しても同様な性質が成り立つことを Dekker [5] が示した．はじめに，浮動小数点数を無誤差で二つの浮動小数点数に分割する計算法を **Algorithm 3** に示す．
ここで，x_h と x_l は，それぞれ x の仮数部上位半分の 26 ビットと下位半分の 26 ビットに対応するものである．ただし，必ずしもビットパターンが一致するわけではない．この式は，x を x_h と x_l の和に誤差なしで変換できることを意味している．

Algorithm 3 を用いて Dekker は無誤差の乗算法 **Algorithm 4** を提案した．

Algorithm 1 や **Algorithm 4** によって，2 つの浮動小数点数 a, b の加算や乗算は，いずれも，その近似 x とその誤差 y という 2 つの浮動小数点数の和 $x + y$ の形に誤差なく変換できることを分かった．このような変換を，無誤差変換（Error-Free Transformations）と呼ぶ．

Algorithm 3 無誤差の分割 Split

浮動小数点数 x を無誤差で二つに分割する計算法．このとき

$$x = x_h + x_l \tag{6}$$

$$|x_h| \geqq |x_l| \tag{7}$$

が成立する．

function $[x_h, x_l] = \mathrm{Split}(x)$
  $c = fl((2^{27}+1)\cdot x)$
  $x_h = fl(c-(c-x))$
  $x_l = fl(x-x_h)$
end

Algorithm 4 乗算の無誤差変換 TwoProduct

乗算に関する無誤差変換．このとき

$$a \times b = x + y \tag{8}$$

$$|y| \leqq \frac{1}{2}\,\mathrm{ulp}(x) \tag{9}$$

が成立する．

function $[x, y] = \mathrm{TwoProduct}(a, b)$
  $x = fl(a \cdot b)$
  $[a_h, a_l] = \mathrm{Split}(a)$
  $[b_h, b_l] = \mathrm{Split}(b)$
  $y = fl(a_l \cdot b_l - (((x - a_h \cdot b_h) - a_l \cdot b_h) - a_h \cdot b_l))$
end

3 DD の誤差解析

3.1 DD とは

倍精度数を用いた四倍精度数の表現方法に付いて述べる．本論文を通して倍精度浮動小数点数をふたつ用いて構成される四倍精度の数を DD 数と呼ぶ．$a_h, a_l \in \mathbb{F}$ とするとき，DD 数 a は

$$a = a_h + a_l \tag{10}$$

$$|a_l| \leq \frac{1}{2}\,\mathrm{ulp}(a_h) \tag{11}$$

と書ける．

3.2 DD の四則演算法

3.2.1 加減算

DD 数の加減算法として二つの手法が知られている．ひとつは `ieee_add` であり，もうひとつは `cray_add` である．`ieee_add` は TwoSum を二回用いて高精度な結果を返すアルゴリズムであり，`cray_add` はそのアルゴリズムを簡潔にしたアルゴリズムである．本報告では `cray_add` について解析を行なう．

Algorithm 5 の誤差上限に関して次の定理が成立する．

Theorem 1
DD 数 $x = x_h + x_l, y = y_h + y_l$ の加算法 **Algorithm 5** の誤差上限は，

$$|z - (x+y)| \leqq 2^{-104} fl(|x_h| + |y_h|) \tag{13}$$

と書ける．

Algorithm 5 cray_add

倍精度演算に基づく四倍精度加算法．このとき

$$z_h + z_l \simeq (x_h + x_l) + (y_h + y_l) \tag{12}$$

が成立する．

function $z = \text{cray\_add}(x, y)$
  $[z_h, z_l] = \text{TwoSum}(x_h, y_h)$
  $z_l = fl(z_l + x_l + y_l)$
  $[z_h, z_l] = \text{FastTwoSum}(z_h, z_l)$
end

proof.
Algorithm 5 を等価な以下の **Algorithm 6** に変換して考える．

Algorithm 6 cray_add (**Algorithm 5**) と等価な計算法

function $z = \text{cray\_add}(x, y)$
  $[t_1, t_2] = \text{TwoSum}(x_h, y_h)$
  $t_3 = fl(x_l + y_l)$
  $t_4 = fl(t_2 + t_3)$
  $[z_h, z_l] = \text{FastTwoSum}(t_1, t_4)$
end

ここで， TwoSum や FastTwoSum は無誤差なので次式が成立する：

$$z_h + z_l = t_1 + t_4 \tag{14}$$
$$t_1 + t_2 = x_h + y_h \tag{15}$$

ここで，**Algorithm 6** で発生する誤差 e は

$$\begin{aligned}
e &= |z - (x + y)| && (16)\\
&= |z_h + z_l - (x_h + x_l + y_h + y_l)| && (17)\\
&= |t_1 + t_4 - (t_1 + t_2 + x_l + y_l)| \qquad \because (14), (15) && (18)\\
&= |t_4 - (t_2 + x_l + y_l)| && (19)\\
&= |t_4 - fl(t_2 + t_3) + fl(t_2 + t_3) - ((t_2 + t_3) - t_3 + x_l + y_l)| && (20)\\
&= |fl(t_2 + t_3) - (t_2 + t_3) + fl(x_l + y_l) - (x_l + y_l)| && (21)\\
&\leqq |fl(t_2 + t_3) - (t_2 + t_3)| + |fl(x_l + y_l) - (x_l + y_l)| && (22)
\end{aligned}$$

と書ける．すなわちこれは t_3 と t_4 の丸め誤差が **Algorithm 6** の誤差になることを示している．

ここで t_3 の丸め誤差は

$$|fl(x_l + y_l) - (x_l + y_l)| \leqq \text{eps}\,(|x_l| + |y_l|) \leqq \text{eps}^2\,(|x_h| + |y_h|) \tag{23}$$

と書ける．同様に t_4 の丸め誤差は

$$|fl(t_2 + t_3) - (t_2 + t_3)| \leqq \text{eps}\,(|t_2| + |t_3|) \tag{24}$$

と書け，$|t_2|, |t_3|$ はそれぞれ，

$$|t_2| \leqq \text{eps}\, fl(|x_h| + |y_h|) \leqq (1 + \text{eps})\ \text{eps}\,(|x_h| + |y_h|) \tag{25}$$
$$|t_3| \leqq (1 + \text{eps})\,(|x_l| + |y_l|) \leqq (1 + \text{eps})\ \text{eps}\,(|x_h| + |y_h|) \tag{26}$$

と書けるので，**Algorithm 6** の誤差上限は

$$e \leqq \text{eps}^2\,(|x_h| + |y_h|) + 2\,\text{eps}^2\,(1 + \text{eps})\,(|x_h| + |y_h|) \tag{27}$$
$$\leqq \text{eps}^2\,(3 + 2\,\text{eps})\,(|x_h| + |y_h|) \tag{28}$$

と書ける．$fl(|x_h| + |y_h|)$ を用いて書き直すと，

$$e \leqq \text{eps}^2\,(3 + 2\,\text{eps})\,(|x_h| + |y_h|) \tag{29}$$
$$\leqq \text{eps}^2\,(3 + 2\,\text{eps})\,(1 + \text{eps})\, fl(|x_h| + |y_h|) \tag{30}$$
$$\leqq 2^{-104}\, fl(|x_h| + |y_h|) \tag{31}$$

と書け所望の結果が得られた． □

3.2.2 乗算

Hida らによる DD 数の乗算法を示す（**Algorithm 7**）．

Algorithm 7 mul

倍精度演算に基づく四倍精度乗算法．このとき

$$z_h + z_l \simeq (x_h + x_l) \cdot (y_h + y_l) \tag{32}$$

が成立する．

function $z = \text{mul}(x, y)$
  $[z_h, z_l] = \text{TwoProduct}(x_h, y_h)$
  $z_l = fl(z_l + x_h \cdot y_l + x_l \cdot y_h)$
  $[z_h, z_l] = \text{FastTwoSum}(z_h, z_l)$
end

Algorithm 7 の誤差上限に関して次の定理が成立する．

Theorem 2
DD 数 $x = x_h + x_l, y = y_h + y_l$ の乗算法 **Algorithm 7** の誤差上限は,

$$|z - x \cdot y| \leqq 2^{-102} |fl(x_h \cdot y_h)| \tag{33}$$

と書ける.

proof.
Algorithm 7 を等価な **Algorithm 8** に変換して考える.

Algorithm 8 mul (**Algorithm 7**) と等価な計算法

```
function z = mul(x, y)
  [t1, t2] = TwoProduct(xh, yh)
  t3 = fl(xh · yl)
  t4 = fl(xl · yh)
  t5 = fl(t3 + t4)
  t6 = fl(t2 + t5)
  [zh, zl] = FastTwoSum(t1, t6)
end
```

ここで, TwoProduct や FastTwoSum は無誤差なので次式が成立する :

$$z_h + z_l = t_1 + t_6 \tag{34}$$
$$t_1 + t_2 = x_h \cdot y_h \tag{35}$$

ここで, このアルゴリズムで発生する誤差 e は

$$\begin{aligned}
e &= |z - x \cdot y| && (36)\\
&= |z_h + z_l - (x_h + x_l) \cdot (y_h + y_l)| && (37)\\
&= |t_1 + t_6 - (x_h \cdot y_h + x_h \cdot y_l + x_l \cdot y_h + x_l \cdot y_l)| \quad \because (34) && (38)\\
&= |t_1 + t_6 - (t_1 + t_2 + x_h \cdot y_l + x_l \cdot y_h + x_l \cdot y_l)| \quad \because (35) && (39)\\
&= |fl(t_2 + t_5) - (t_2 + t_5) + t_5 - (x_h \cdot y_l + x_l \cdot y_h + x_l \cdot y_l)| && (40)\\
&= |fl(t_2 + t_5) - (t_2 + t_5) + fl(t_3 + t_4) - (t_3 + t_4) + (t_3 + t_4) - (x_h \cdot y_l + x_l \cdot y_h + x_l \cdot y_l)| && (41)\\
&\leqq |fl(t_2 + t_5) - (t_2 + t_5)| + |fl(t_3 + t_4) - (t_3 + t_4)| + |fl(x_h \cdot y_l) - x_h \cdot y_l| \\
&\qquad + |fl(x_l \cdot y_h) - x_l \cdot y_h| + |x_l \cdot y_l| && (42)
\end{aligned}$$

と書ける.
ここで t_3, t_4 の丸め誤差は

$$|fl(x_h \cdot y_l) - (x_h \cdot y_l)| \leqq \text{eps}\,|x_h|\,|y_l| \leqq \text{eps}^2\,|x_h|\,|y_h| \tag{43}$$
$$|fl(x_l \cdot y_h) - (x_l \cdot y_h)| \leqq \text{eps}\,|x_l|\,|y_h| \leqq \text{eps}^2\,|x_h|\,|y_h| \tag{44}$$

と書ける.

t_5 の丸め誤差は

$$|fl(t_3+t_4)-(t_3+t_4)| \leqq \text{eps}\,(|t_3|+|t_4|) \tag{45}$$

と書け, $|t_3|, |t_4|$ はそれぞれ,

$$|t_3| \leqq (1+\text{eps})\,|x_h|\,|y_l| \leqq (1+\text{eps})\,\text{eps}\,|x_h|\,|y_h| \tag{46}$$
$$|t_4| \leqq (1+\text{eps})\,|x_l|\,|y_h| \leqq (1+\text{eps})\,\text{eps}\,|x_h|\,|y_h| \tag{47}$$

と書けるので t_5 の丸め誤差は

$$|fl(t_3+t_4)-(t_3+t_4)| \leqq 2(1+\text{eps})\,\text{eps}^2\,|x_h|\,|y_h| \tag{48}$$

となる.

t_6 の丸め誤差は

$$|fl(t_2+t_5)-(t_2+t_5)| \leqq \text{eps}\,(|t_2|+|t_5|) \tag{49}$$

と書け, $|t_2|, |t_5|$ はそれぞれ,

$$|t_2| \leqq \text{eps}\,fl(|x_h|\,|y_h|) \leqq \text{eps}\,(1+\text{eps})\,|x_h|\,|y_h| \tag{50}$$
$$|t_5| \leqq (1+\text{eps})\,(|t_3|+|t_4|) \leqq 2(1+\text{eps})^2\,\text{eps}\,|x_h|\,|y_h| \tag{51}$$

と書けるので t_6 の丸め誤差は

$$|fl(t_2+t_5)-(t_2+t_5)| \leqq (1+\text{eps})\,(3+2\,\text{eps})\,\text{eps}^2\,|x_h|\,|y_h| \tag{52}$$

となる.

$|x_l y_l|$ の上限は,

$$|x_l y_l| \leqq \text{eps}^2\,|x_h|\,|y_h| \tag{53}$$

と書ける.

以上より **Algorithm 8** の誤差上限は

$$\begin{aligned} e &\leqq \text{eps}^2\,|x_h|\,|y_h| + \text{eps}^2\,|x_h|\,|y_h| + 2(1+\text{eps})\,\text{eps}^2\,|x_h|\,|y_h| \\ &\qquad + (1+\text{eps})\,(3+2\,\text{eps})\,\text{eps}^2\,|x_h|\,|y_h| + \text{eps}^2\,|x_h|\,|y_h| \qquad (54) \\ &\leqq \left(8+7\,\text{eps}+2\,\text{eps}^2\right)\,\text{eps}^2\,|x_h|\,|y_h| \qquad (55) \end{aligned}$$

と書ける. ここで誤差上限を $fl(x_h \cdot y_h)$ を用いて書き直すと,

$$\begin{aligned} e &\leqq \left(8+7\,\text{eps}+2\,\text{eps}^2\right)\,\text{eps}^2\,(1+\text{eps})\,fl(|x_h|\,|y_h|) \qquad (56) \\ &\leqq \left(8+15\,\text{eps}+9\,\text{eps}^2+2\,\text{eps}^3\right)\,\text{eps}^2\,|fl(x_h \cdot y_h)| \qquad (57) \\ &\leqq 2^{-102}\,|fl(x_h \cdot y_h)| \qquad (58) \end{aligned}$$

と書け所望の結果が得られた. □

3.2.3 除算

Hida らは彼らの QD/DD の中で DD 数の除算法を二つ提案している．ひとつは sloppy_div であり，もうひとつは accurate_div である．accurate_div は DD 数の乗算法を用いて高精度な結果を返すアルゴリズムであり，sloppy_div はそのアルゴリズムを簡潔にしたアルゴリズムである．その他，山中・大石が提案した fast_div という高速な計算法が存在する．fast_div は，sloppy_div や accurate_div で用いている DD 数の乗算法を用いていないため，高速に実行できるという特徴がある．本報告では実行時間に重点を置き fast_div について述べる．

Algorithm 9 fast_div

倍精度演算に基づく四倍精度除算法．このとき

$$z_h + z_l \simeq \frac{x_h + x_l}{y_h + y_l} \tag{59}$$

が成立する．

function $z = \text{fast\_div}(x,\ y)$
$y_r = fl(1/y_r)$
$z_h = fl(x_h \cdot y_r)$
$[t_h, t_l] = \text{TwoProduct}(z_h, y_h)$
$z_l = fl(((x_h - t_h) - t_l) \cdot y_r + z_h \cdot (x_l/x_h - y_l \cdot y_r))$
$[z_h, z_l] = \text{FastTwoSum}(z_h, z_l)$
end

Algorithm 9 は次式に基づく．

$$\frac{x_h + x_l}{y_h + y_l} = \frac{x_h(1+x_r)}{y_h(1+y_r)} = \frac{x_h}{y_h}(1+x_r)\left(1 - y_r + \frac{y_r^2}{1+y_r}\right) \simeq \frac{x_h}{y_h} + \frac{x_h}{y_h}\left(x_r - y_r + \mathcal{O}\left(\text{eps}^2\right)\right) \tag{60}$$

ここで，x_r, y_r はそれぞれ，

$$x_r = \frac{x_l}{x_h}, \qquad y_r = \frac{y_l}{y_h} \tag{61}$$

であり，これらは DD 数の定義より

$$|x_r| \leqq \text{eps}, \qquad |y_r| \leqq \text{eps} \tag{62}$$

を満たす．なお，**Algorithm 9** の誤差上限は本報告では割愛する．

4 高精度・高信頼・高可搬な指数関数計算手法

4.1 既存の高信頼初等関数計算法

倍精度数を利用し，発生する全ての誤差を考慮した手法として，Muller らが提案した手法 [6] や Rump が提案した手法 [7] などがある．Muller らの手法は大変洗練されており，浮動小数点数倍精度のほぼ全ての入力に対し，最近点の値を返す．これらは CRlibm ライブラリ [8] として利用できる．

ただし，CRlibm のそれぞれの関数に対する実装は少々難解で，CRlibm で実装されている形式以外に移植することは困難を極める．Rump の手法は，事前にいくつかの値に対し高精度な値を保存しておき，それらを利用し精度保証された区間を計算することができる．最大で 6ulp の相対誤差が混入する可能性がある．Rump の手法により出力される区間の相対誤差は入力に依存するが，上端下端型区間を用いて出力されるため，特別な場合を除き理論上 2ulp を下回ることはない．

近年，整数のみを利用し擬似的に浮動小数点数を形成することで任意に精度を変化させることのできる任意精度浮動小数点システムを用いたライブラリが製作されている (Hanrot らによる MPFR など [1])．MPFR では多倍長計算を行なえる利点を生かし，初等関数の計算を精度保証付きで高精度に行なうことができる．ただし，倍精度浮動小数点数のみを利用した手法と比べて実行時間が遅くなる傾向がある．

4.2 提案手法

本報告では浮動小数点数を用いた高精度・高信頼・高可搬な指数関数計算法について述べる．$x \in \mathbb{F}$ を満たす全ての浮動小数点数 x は次のように表せる．

$$x = (-1)^s \sum_{i=0}^{n-1} m_i \cdot 2^{e-i} \tag{63}$$

なお，ここで s は符号部，e は指数部，m_i は仮数部であり 0 または 1 である（ただし $m_0 = 1$）．また IEEE 754-1985 標準規格に従う倍精度浮動小数点形式 ならば $n = 53$ である．このとき，正の浮動小数点数に対する指数関数値は

$$\exp(x) = \exp\left(\sum_{i=0}^{n-1} m_i \cdot 2^{e-i}\right) = \prod_{i=0}^{n-1} \left(\exp(2^{e-i})\right)^{m_i}$$

と書け，負の浮動小数点数に対しては

$$\exp(x) = 1/\prod_{i=0}^{n-1} \left(\exp(2^{e-i})\right)^{m_i}$$

と書ける．

ここで，$\exp(2^i)$ の値を DD 数として事前に計算し，保存しておくことを考える．すなわち，DD 数 $p^{(i)}$ に対して，浮動小数点数 $p_h^{(i)}, p_l^{(i)} \in \mathbb{F}$ が

$$\left|\exp(2^i) - \left(p_h^{(i)} + p_l^{(i)}\right)\right| \leqq \mathrm{eps}^2 \exp(2^i) \tag{64}$$

を満たすならば，正の浮動小数点数に対する指数関数値は

$$\exp(x) \simeq \prod_{i=0}^{n} \left(p_h^{(i)} + p_l^{(i)}\right)^{m_i}$$

と書ける．

DD 数を **Algorithm 7** を用いて n 個掛け合わせるアルゴリズムを **Algorithm 10** に示す：

Algorithm 10 muls

Algorithm 7 を用いた四倍精度乗算法．このとき

$$q_h + q_l \simeq \prod_{i=0}^{n-1} (p_h(i) + p_l(i)) \tag{65}$$

が成立する．

```
function  q = muls(p(0 : n − 1))
  q = p(0);
  for i = 1 : n − 1
    q = mul(p(i), q);
  end
```

Theorem 3

Algorithm 10 の誤差上限は，$n \leqq 2^6$ のとき，以下のように書ける：

$$|\exp(x) - \mathbf{muls}(x)| \leqq (11n - 10)\ eps^2 \exp(x). \tag{66}$$

式 (63) は全ビットを 1 ビット毎に分割したが，高速化を狙い，数ビットまとめて処理が行なえる．例えば，以下の実行結果は一度に 6 ビットの処理を行なった際の結果である．実行結果よりこの手法は，高精度・高信頼・高可搬かつ高速な指数関数計算手法であると言える．

1. 四倍精度区間の出力

計算手法	最大誤差	実行時間比
提案手法	3.2×10^{-30}	0.027
MPFR	1.2×10^{-32}	(1.000)

※ MPFR は 106 bit に設定して計算．

2. 倍精度区間の出力（入力 10,000 点の乱数，単位 ulp）

計算手法	最小誤差	平均	最大誤差	時間比
提案手法	0.000078	0.251	0.499	3.1
CRlibm	0.500	0.500	0.500	(1.0)
Intlab	1.000	1.268	2.000	–

参考文献

[1] The GNU MPFR Library: http://www.mpfr.org/

[2] D. H. Bailey, Y. Hida, X. S. Li and B. Thompson. "ARPREC: an arbitrary precision computational package", Lawrence Berkeley National Laboratory. Berkeley. CA94720. 2002.

[3] Y. Hida, X. S. Li and D. H. Bailey. "Quad-Double Arithmetic: Algorithms, Implementation, and Application" October 30, 2000 Report LBL-46996.

[4] D. E. Knuth: The Art of Computer Programming: Seminumerical Algorithms, vol. 2, Addison-Wesley, Reading, Massachusetts, 1969.

[5] T. J. Dekker: A floating-point technique for extending the available precision, Numer. Math., 18 (1971), pp. 224–242.

[6] J.M.Muller, Elementary Functions, Birkhauser Boston, 2nd edition, 2006.

[7] S.M.Rump, Rigorous and portable standard functions. BIT Numerical Mathematics, 41(3), pp. 540–562, 2001.

[8] CRlibm – Correctly Rounded mathematical library: http://lipforge.ens-lyon.fr/www/crlibm/